## 1.5. A Short Example

Before diving into the details of iterative development, requirements analysis, UML, and OOA/D, this section presents a bird's-eye view of a few key steps and diagrams, using a simple example—a "**dice game**" in which software simulates a player rolling two dice. If the total is seven, they win; otherwise, they lose.

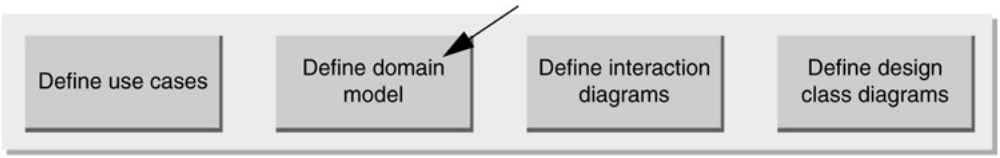
### Define Use Cases


Requirements analysis may include stories or scenarios of how people use the application; these can be written as use cases.

Use cases are not an object-oriented artifact—they are simply written stories. However, they are a popular tool in requirements analysis. For example, here is a brief version of the Play a Dice Game **use case**:

> **Play a Dice Game**: Player requests to roll the dice. System presents results: If the dice face value totals seven, player wins; otherwise, player loses.

### Define a Domain Model




Object-oriented analysis is concerned with creating a description of the domain from the perspective of objects. There is an identification of the *concepts*, *attributes*, and *associations* that are considered noteworthy.

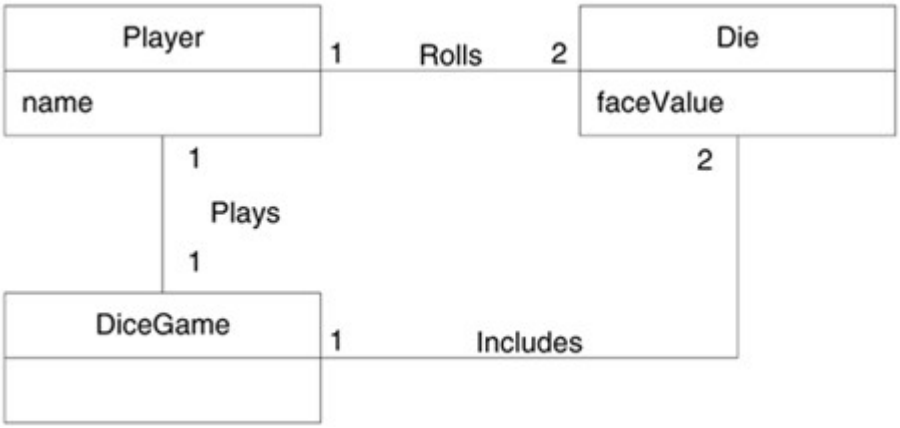
The result can be expressed in a domain model that shows the noteworthy domain concepts or objects.

---

From: Craig Larman: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development,
(3rd Edition), Prentice Hall, 2004, Print ISBN-10: 0-13-148906-2,
Print ISBN-13: 978-0-13-148906-6

For example, a partial domain model is shown in Figure 1.3.

***Figure 1.3. Partial domain model of the dice game.***




This model illustrates the noteworthy concepts Player, Die, and DiceGame, with their associations and attributes.

Note that a domain model is **not a description of software objects**; it is a **visualization of** the **concepts or mental models of** a **real-world** domain. Thus, it has also been called a **conceptual** object **model**.

## Assign Object Responsibilities and Draw Interaction Diagrams

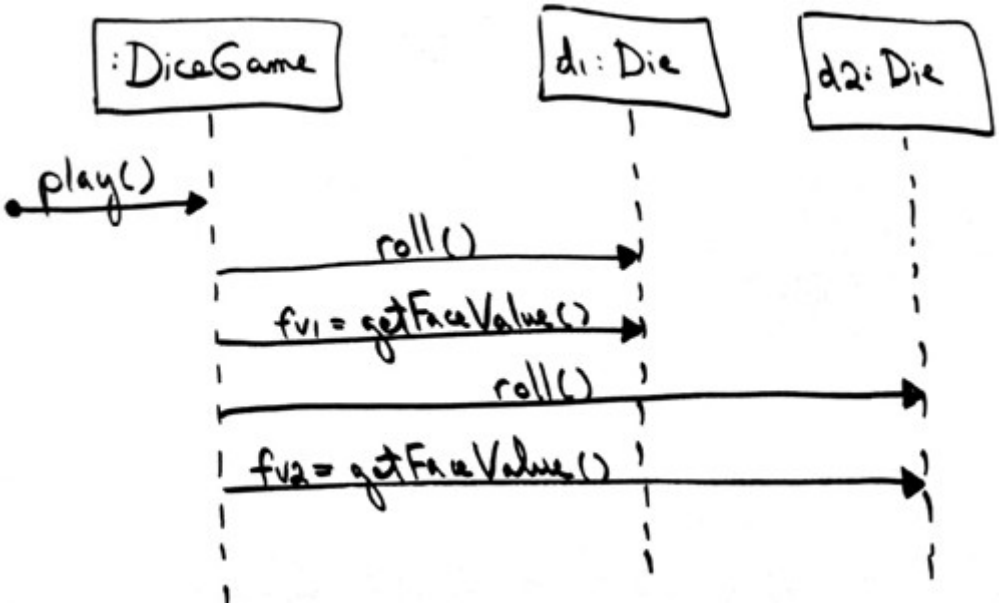

Object-oriented design is concerned with defining software objects—their *responsibilities* and *collaborations*. A common notation to illustrate these collaborations is the sequence diagram (a kind of UML interaction diagram). It shows the flow of messages between software objects, and thus the invocation of methods.

For example, the sequence diagram in Figure 1.4 illustrates an OO software design, by sending messages to instances of the DiceGame and Die classes. Note this illustrates a common real-world way the UML is applied: by sketching on a whiteboard.

---

From: Craig Larman: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development,
(3rd Edition), Prentice Hall, 2004, Print ISBN-10: 0-13-148906-2,
Print ISBN-13: 978-0-13-148906-6

*Figure 1.4. Sequence diagram illustrating messages between software objects.*



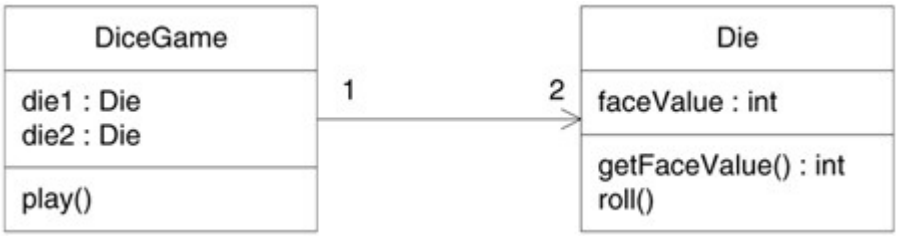

Notice that although in the real world a player rolls the dice, in the software design the DiceGame object "rolls" the dice (that is, sends messages to Die objects). Software object designs and programs do take some inspiration from real-world domains, but they are not direct models or simulations of the real world.

## Define Design Class Diagrams


In addition to a dynamic view of collaborating objects shown in interaction diagrams, a static view of the class definitions is usefully shown with a design class diagram. This illustrates the attributes and methods of the classes.

For example, in the dice game, an inspection of the sequence diagram leads to the partial design class diagram shown in Figure 1.5. Since a play message is sent to a DiceGame object, the DiceGame class requires a play method, while class Die requires a roll and getFaceValue method.

*Figure 1.5. Partial design class diagram.*




In contrast to the domain model showing real-world classes, this diagram shows software classes.

Notice that although this design class diagram is not the same as the domain model, some class names and content are similar. In this way, OO designs and languages can support a lower representational gap between the software components and our mental models of a domain. That improves comprehension.

---

From: Craig Larman: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development,
(3rd Edition), Prentice Hall, 2004, Print ISBN-10: 0-13-148906-2,
Print ISBN-13: 978-0-13-148906-6